The Free Speech Engine: Conversational Web Service Compatibility for Free

Stegers, R.G.M.; van Harmelen, F.A.H.; ten Teije, A.C.M.

published in
Proceedings of the 2009 International Conference on Semantic Web & Web Services, SWWS 2009
2009

document version
Early version, also known as pre-print

The Free Speech Engine
Conversational web service compatibility for free

Ruud Stegers, Frank van Harmelen and Annette ten Teije
Department of computer science, VU University Amsterdam, The Netherlands
{rstegers},{frankh},{annette}@few.vu.nl

Abstract— *With the emergence of a ubiquitous web of data and services, interoperability between those services without the need for pre-coordination becomes of great importance. However, current web services are often engineered in the remote function call style. This imposes very specific interaction patterns on the peers involved, and creates a significant barrier to interoperability.*

This work introduces a different communication abstraction, the Free Speech system, which aims removing any ordering constraints on the communication that are not strictly needed for business requirements.

We compare our first implementation of some services in the Free Speech system with a classic WSDL implementation of the same services. Our first experiments show that the Free Speech approach results in a much higher degree of interoperability between services with different business requirements.

Keywords: Web services, interoperability, protocol

1. Introduction

Generally accessible web services have been around now for a while, providing a variety of services to clients. Different technologies are being used to facilitate the communication and coordination between the peers involved. However, without exception these technologies are used to define very specific interaction patterns between the peers involved. The result is a huge amount of clients and services which might be compatible on the level of the functional business requirements, but which are not due to interoperability issues on the communication level. In the light of creating an open web of (compatible) services, this is an undesirable situation.

Two levels of coordination and communication may be discriminated. The first is the overall workflow. On this level the ordering of tasks is determined. These tasks are generally high-level and are determined by the domain (e.g. a payment-task). On the lower level there is communication and coordination between the peers to execute a single task (e.g. exchanging account numbers). This work introduces a communication model and interaction framework currently focusing on the lowest level, although in the future it will be extended to encompass the also the workflow.

Protocol definitions are a balance between simplicity and semantic clarity on the one hand, and freedom to suit the specific needs of a peer on the other hand. The more freedom is allowed the more likely an existing protocol is to meet the requirements of an application, at the cost of a higher implementation effort in general. The aim should be to be as restrictive as possible without compromising on the areas where room is required for business specific choices. The benefit of such an approach is that the restrictive part provides clarity on the semantics of those parts of the interaction, which in turn is a strong foundation to allow flexibility and freedom on other parts without loosing clarity.

Current web services protocols generally lack flexibility on the interaction level: the order of messages or calls is strictly prescribed, thereby leaving no room for business specific adjustments. The framework introduced here, the Free Speech protocol, allows full flexibility regarding business requirements by dropping all ordering constraints which do not stem from business requirements, while at the same time maintaining semantics clarity and preserving compatibility between services with the same functionality.

In section 2 existing technologies in the area of high level communication on the Internet will be reviewed. Web services, agents and tuple-space based concepts will be considered. Their strong and weak points in relation to the goal of reaching an open web op services will be evaluated. Section 3 will introduce a new communication and coordination concept, the Free Speech protocol, followed by an evaluation of a use case in section 4. In this section the interaction between two clients and two servers will be evaluated. The clients and servers share the domain, but have different business requirements. A comparison will be made between an implementation based on a WSDL interface and a prototype implementation of the newly introduces system, called the Free Speech Engine. Finally in section 5 the experiment will be evaluated followed by conclusions.

2. Existing technologies

2.1 The remote function call style

Most of currently available web services employ a 'remote function call' style (RFC) of communication. Whether this is build on RPC (Remote Procedure Calls), Java RMI (Remote Method Invocation), a form of REST (Representational State Transfer) or a WSDL implementation on top of some low-level protocol, the principle is the same: data is send over the network containing a function identifier in some form and a set of parameters applicable to this function [10], [8], [5], [1]. The service publishes the supported interface in some form (machine

or human readable) and expects the input to be conform the specification. If case multiple subsequent calls are required to achieve the final result, the allowed sequences are also published, also here in either human or machine readable form.

Compared to pure message based communication the remote function call style is an abstraction with certain advantages. The biggest advantage is that it is very close to the normal way of thinking of a programmer. Additionally the semantics are clearly specified and the communication itself is efficient. Unfortunately the choice for this abstraction does come with a price: a strict interaction requirement is part of the interface. A function based interface dictates a specific sub-division of the overall task in subtasks. Since there is a one-to-many relation between the overall functionality and the possible subdivisions in subtasks, two *independently developed* interfaces will most certainly be incompatible. This inherent incompatibility makes the RFC style of communication hard to maintain in an open network of services like the web.

Given that many interface-incompatible services exist, one solution direction explored now is to add semantic annotations to the service specifications (WSMO, WSDL-S, OWL-S) [9], [4], [3]. The idea is to reason about the differences and by doing so being able to overcome these differences. Although data mappings may be within reach for a set of services, solving incompatibilities on the interaction level is still far away. Much of the problem is caused by the choice of communication abstraction. While on the long term semantic techniques may help in solving part of the integration problem, in this work an effort will be made instead to avoid the problem at all by choosing another abstraction.

2.2 Agents

Although agents and web services differ quite a bit in their design goals, it is interesting to take a look at the way agents interact. On the level of communication, agents are different in the sense that they generally do not use the remote procedure call paradigm. Instead messages are exchanged regarding *facts about the world* which are interpreted and consumed by the receiving agent. The facts are encapsulated in a wrapper of performatives that define the intention of the sender in relation to the content.

Agent Communication Languages (ACL) date back to the early 90's with KQML. Since then many dialect and variant have been developed, among which FIPA-ACL [2]. The semantics of the performatives in FIPA-ACL are described in terms of a BDI (Belief, Desire and Intention) framework regarding preconditions and expected result. For example, *inform* states that the sender believes the encapsulated fact, does not believe the receiver to have knowledge about the fact, but does intend the receiver to know it. Although BDI is used widely in agents, non-BDI based agents do exist.

The main advantage of using an ACL is that the content of the message is given meta-semantics concerning the senders intention. This potentially allows an unlimited set of conversations and is - in that sense - truly open. However, the content itself must also have clear semantics especially in relation to the applied performative: wrapping a question inside the earlier described *inform* doesn't make sense. Since this is not generally forbidden, messages with doubtful semantics may be encountered and must be dealt with.

The agent model does not prescribe anything particular about the internals of the agent: the agent is autonomous and the internal domain models may differ. Although this is considered a feature, it does complicate things on the level of the interaction between the content of the message, the associated performative and the internal state of the agent: it is not always unambiguously clear what the effect of a message (or a sequence of messages) will be on the receiving peer. While for agents this may work, for services it poses a risk. For many (business) services uncertainty regarding the final effect of the communication is not acceptable: it must be very clear what has been (dis)agreed on.

Although the openness of the communication offered by the use of performatives is attractive, the underlying assumptions seem to be incompatible with the goals of web services. Major adjustments are required to get to the level of semantic certainty that is required for use in business web services. Not only should the BDI model be exchanged for something more deterministic, but also regarding the allowed content of the messages restrictions are necessary. These considerations will be taken into account in the proposed protocol, while holding on to the concept of adding additional semantics to the communication by the use of performatives.

2.3 Tuple and triple spaces

Mid eighties the TupleSpace Coordination paradigm was introduced with a framework called Linda [6]. Instead of relying on point-to-point communication, a conceptual shared space was created to which all the participants would have access. The central means for data exchange is a tuple, which is a vector containing a number of (data) fields. By reading and writing tuples to and from the shared space participants would be able to exchange data and work together without knowing each other and while being spatially and temporally disconnected. The Linda API extended existing languages with simple primitives which amongst others allowed reading of tuples based on a template so the process could wait for a tuple with a specific format to be posted. For example: $rd\,\langle Temperature, Amsterdam, ?\rangle$ would yield a tuple (if available) in which the first two field were exactly matched, and the third field was filled in.

Although effective in closed environments, Linda (and alike) are limited to syntactic matching of tuples. Therefore all the peers need to agree on the syntax and semantics of the tuples in the space (e.g. the temperature being in degrees centigrade). This requirement is hard to meet in open environments like the current web. In contrast, in the closed environment of a parallel cluster Linda may

be used successfully for inter process coordination since agreement on syntax and semantics is not an issue.

Since the major benefits of TupleSpace coordination are still very relevant (the spatial and temporal independence of the peers), more recent work has focused on using the ideas of TupleSpaces with modern semantic technologies. The TripCom project is one of those efforts which has lead to an implementation of a TripleSpace [11], [7], [12]. The TripleSpace stores RDF triples instead of general tuples. Additional primitive are added to allow semantic matching and retrieval of subgraphs of triples based on their semantic coherence (e.g. the ability to execute SPARQL queries on a space).

Although bringing semantics to the TupleSpace is an important step, it is not enough to solve the coordination problem. The RDF does decrease the dependencies betweens peers on the content level, but to avoid being stuck with only Publish-Consume interactions an additional step is required: a more sophisticated coordination layer must be added to build upon, without loosing the inherent advantages of the triple spaces.

For the solution presented, the idea of a shared space that provides spatial and temporal independence will be used and taken further on step by actually adding an actual coordination layer.

3. Service coordination in a shared space

Communication between services is all about agreeing on the details of one or more tasks and possibly returning a result (e.g. come to agree on which book to send, and what the price will be). Therefore protocols should be aimed entirely on reaching the above mentioned agreement.

Given the goal in the open web that as many services as possible should be able to interoperate, and taking into account the open nature of the web where there is a limited tendency to adopt domain specific standards, an integrated communication and coordination protocol is required on top of which the required functionality can be built. Such a protocol must specify, restrict and guide the communication as much a possible (for clarity reasons) without posing restrictions on business specific requirements at the application level. As argued before, RFC style protocols and current agent languages do not satisfy the requirements. Neither does a TripleSpace by itself solve the problem.

In a protocol clear semantics are essential: the protocol definition needs not only to be clear on the semantics of individual messages, but also on the semantics of a sequence of messages. Every sequence of messages must have unambiguous semantics regarding what has been (dis)agreed on.

In the next section a protocol will be proposed which meets the above stated requirements. As stated earlier the higher level workflow perspective will be ignored in this article, and focus will be on the communication concerning one task.

3.1 Shaping data

While the semantics of a concept may be clear, the semantics of communicating that same concept may not be that clear. The type of message, earlier messages and the context may affect the exact interpretation. One way to guarantee a consistent interpretation of a sequence of messages is to reduce the influence of history and context as much as possible. Although this may seem hard or overly restrictive, it can be easily be achieved by defining messages that *operate deterministically on a (conceptually) shared graph*. By doing so, the semantics of individual messages are limited to the effect they have on this graph, while the semantics of a sequence of messages applied to some graph is reduced to the semantics of the graph itself. This way the history and context are captured by the graph.

Now, if the graph would be able to contain all the details of the discussed task, then all the ingredients are present to reach agreement on the details of the task just by exchanging messages that change the graph in small steps into its final form where both peers agree on the content. Given the notion of such a shared graph, it becomes easy to abstract away from the messages themselves by considering the interaction as the combined effort of all peers (may be more that two) to shape the initial version of the graph into a satisfactory final description of the task. To support the interaction process, meta relations can be added to the graph to inform other peers of requirements or intentions (e.g. that some piece of the graph must be provided by another peer).

To illustrate this idea consider two people negotiating the sale of a car by filling in an order template on paper. The template contains slots for all the possible options. During the process the content of the template is changed by both the seller and the buyer until they agree on the specs of the car (and the price that comes with that). Simple verbal remarks are used in the process to guide the negotiation process, being equivalent to meta relations in the computer model.

The purpose of introducing such a new protocol is to achieve interoperability between services as a side product of their implementation. While services based on the 'remote function call' principle would be incompatible on the both the vocabulary and the interaction level, the sketched interaction model would reduce the problem to an issue regarding the domain vocabulary (i.e. the domain specific concepts and relations used in the graph). Difference in the domain vocabulary may be solved by applying data mapping techniques, although this is outside of the scope of this work.

For the car sale a different domain vocabulary would mean to have a different template to fill in all the options. However, a template it is, and the process to fill it remains the same. The RFC style real world equivalent on the other hand, would force the buyer to go through a specific set of questions in fixed order to communicate his wishes.

3.2 Free Speech: the shape of the data

The possible form of the templates and the allowed *graph-shaping* operations define the Free Speech protocol. The basic building blocks of the graph are fixed in advance, allowing to unambiguously communicate by molding a graph containing the building blocks. The communication rules and possible operations (with their semantics) are defined by these building blocks. The actual interpretation of the data in the graph is domain specific. The left hand side of figure 1 shows the three type of building block that have been defined so far. With the building blocks a task specific tree can be defined that will serve as a template for all the task specific details. Meta information regarding the communication process may create cross-connections between nodes and thereby making it a graph.

The action node carries the semantics of the task as a whole. Therefore the root of the tree is always an action node. The action may host either structure nodes or value nodes. The structure nodes adds additional depth to the tree to provide a context group for the semantics of the nodes below it. The value nodes are the holes in the template which may be filled in during the interaction.

The type of node determines which operations may be applied to the node. The meta-data is used to guide the interaction process and is a visible effect of applied operations. An example of this is the *ValidationRequest* operation. A peer can apply this operation to a value node, and it results in the addition the *ValidationRequest* meta-data to the node. The newly added data contains a reference to the peer so it is clear which peer applied it. In the same way a peer may approve a value by applying *ValidationApprove* or give approval to the whole tree by applying *Approve* to the top-level action. When both peers approve the root action node, there is agreement and both peers are held to fulfill their part of the action as described by the domain specific semantics.

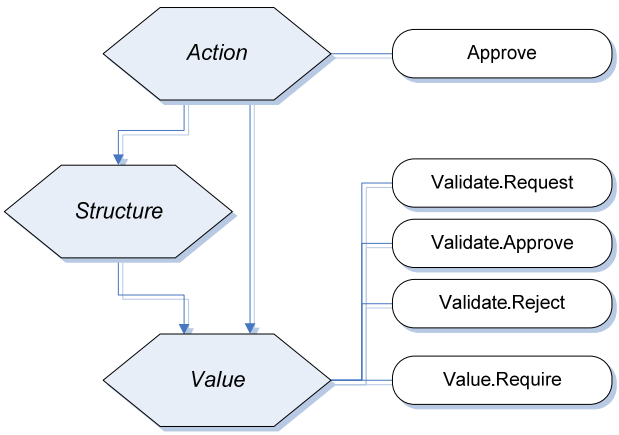

Fig. 1: The basic vocabulary building blocks. The hexagon shapes denote the elementary types. The arrows between them represent a 'may contain' relation. The rounded blocks are operations that may be applied to the type they are connected to.

To summarize the concept: peers will communicate by applying operations to the nodes of a shared graph. The operations are universal and shared between the peers, and so are the basic building blocks of the graph. Both operations and the building blocks are application independent while a vocabulary built using these building blocks is all that is needed to allow for a wide variety of implementations of that application.

4. Use case: A mail client and server

In this section a use case will be considered. The peers from the use case will all be implemented both on top of WSDL and on the Free Speech protocol. Section 4.1 shows the functional specification of two mail clients and two mail servers based on their business requirements. The goal of the Free Speech protocol is to separate the communication and coordination infrastructure from the functionality in such a way that a variety of business requirements can be implemented using the same shared specification (i.e. the domain specific vocabulary) without the need for changes to the specification (retain interface compatibility). The resulting compatibility, despite of the different business requirements will serve as an indicator of the potential of the Free Speech protocol.

4.1 Business requirements

Two mail servers and two mail client will be implemented. The functionality of the mail servers is just what one would expect from such a server: given a from- and to-address, a subject and the content of the mail, the server will compose the message and pass it to the correct SMTP server. The first prototype server will do this regardless of the content. The second implementation will only accept mails for it's own domain and requires a proper sender.

The first mail client is a naive implementation which just passes anything the user provides to the mail server without restrictions. The order in which the data becomes available is unspecified. A more privacy aware client has been added that requires confirmation of the server regarding the destination address: the server needs to approve the address before other mail related content is provided.

Table 1: The functional features based on the business requirements of the selected clients and servers.

Client 1	Passes all information to the server when it comes available.
Client 2	Requires approval for the destination address before offering any of the other email fields to the server.
Server 1	Accept any combination of mail field as long as a destination is given.
Server 2	Accepts only mails to the *cs.vu.nl* domain and requires a proper sender address.

4.2 A WSDL implementation

Based on the business requirements a WSDL interface has been defined. Figure 2 shows the basic SendMail

```
<description ...>
  <types>
    <xs:schema ...>
      <xs:element name="SendMail"
                  type="tSendMail"/>
      <xs:complexType name="tSendMail">
        <xs:sequence>
          <xs:element name="ToAddress"
                      type="xs:string"/>
          <xs:element name="ToDisplayName"
                      type="xs:string"/>
          <xs:element name="FromAddress"
                      type="xs:string"/>
          <xs:element name="FromDisplayName"
                      type="xs:string"/>
          <xs:element name="Subject"
                      type="xs:string"/>
          <xs:element name="Content"
                      type="xs:string"/>
        </xs:sequence>
      </xs:complexType>

    <cs:element name="SendMailSuccessMsg"
                type="xs:string"/>
      <xs:element name="SendMailErrorMsg"
                  type="xs:string"/>
    </xs:schema>
  </types>

  <interface name="SimpleMail">
    <fault name="SendMailError"
           element="SendMailErrorMsg"/>

    <operation name="opSendMail"
               pattern="in-out" ...>
      <input messageLabel="In"
             element="SendMail"/>
      <output messageLabel="Out"
              element="SendMailSuccessMsg"/>
      <outfault ref="SendMailError"
                messageLabel="Out"/>
    </operation>
  </interface>
</description>
```

Fig. 2: A basic WSDL interface for sending mail.

interface. One operation has been defined which carries all the required information in the parameters. When the email has been send successfully a success message will be returned. If an error occurs or if the data is not correct, an error message will follow.

Results. The business requirements of Client 1, Server 1 and Server 2 fit easily on the interface. Client 1 and Server 1 are almost without requirements. Server 2 may use the error message to report back when the given destination address is not acceptable.

For client 2 it is much harder to fit the given business requirements in the existing interface. There is no possibility to send information in separate chunks to the server. Therefor it is impossible to get approval for the given *ToAddress* without already giving the other message details content of the message. If this functionality is required, a new operation must be defined and added to the interface. Obviously, this operation might have been part of the interface in the first place in which case the problem would not exist, but it is easy to come up with some other verification request that has not been anticipated and is thus not yet supported. Although demonstrated here on a WSDL interface, this demonstrates the earlier claimed problem with the RPC style web services.

4.3 The Free Speech implementation

The basis of a service definition in the Free Speech protocol is a shared vocabulary for all peers based on the building blocks introduced in section 3.2. For this use case, this vocabulary is defined later in this section.

The Free Speech protocol is implemented in a prototype of the Free Speech Engine, a communication library for peers. This implementation runs on the *.Net* platform and offers access to the operations that operate on the defined vocabulary. It provides the shared view on the data and takes care of the underlying communication and synchronization. The actual peer implementation is a combination of declarative programming in XML augmented with object oriented code where required.

Vocabulary. The domain specific vocabulary was built using the earlier defined building blocks. The tree serves as a template to reach agreement on all required data for the 'Send Mail' domain. Figure 3 shows the structure. For simplicity sake only the most common email fields have been put in the tree. Obviously for a full fledged vocabulary, all generally relevant fields should be present. In no way there is an obligation to actually use all the fields in the conversation. During the interaction, fields will be manipulated through operations, and will be filled in and changed until both peers agree on the content as a whole.

Invisible in this figure, but present nevertheless, are the role definitions. In this vocabulary two roles have been identified: the client that wants to send a mail (*MailClient*) and a server willing to actually send the message on the Internet (*MailServer*).

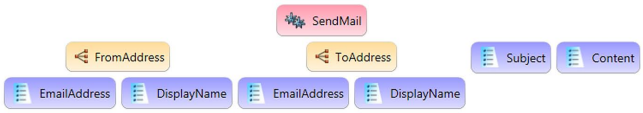

Fig. 3: Generic vocabulary specification for the mail domain. Sendmail is the top-level node of the *Action* type. FromAddress, ToAddress, Subject and Content are child nodes of the action. FromAddress and ToAddress are structure nodes both containing two child nodes of the *Value* type.

The semantics of the structure as a whole follows from the vocabulary description. Next to the description of the tree, there is a human readable explanation of the semantics of the tree. This will provide clear and fixed semantics which serve as a contract and which should be implemented by the programmer (just like with existing web services). These semantics provide the foundation of reliable services based on open interaction patterns while avoiding 'open interpretation'.

The semantics of the in figure 3 presented vocabulary is easy to describe: *The peer that takes the roll of MailServer*

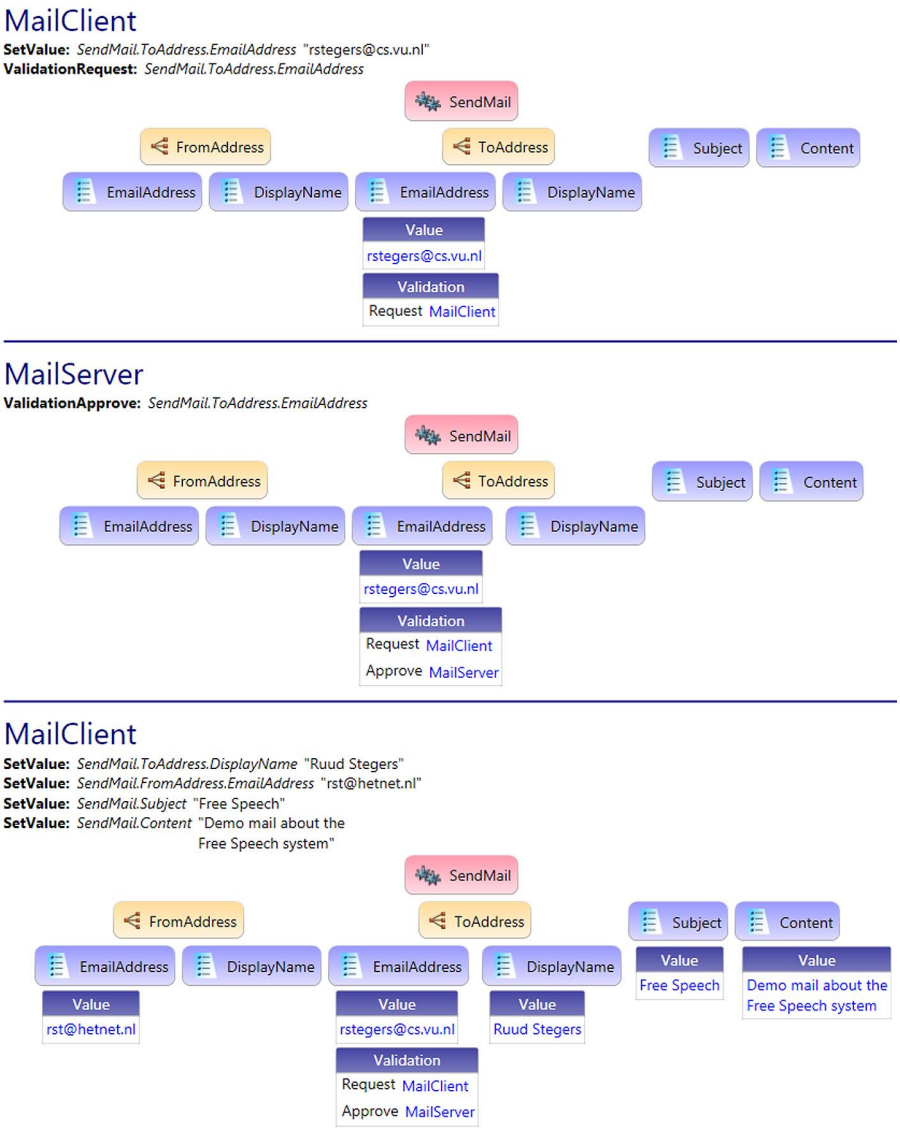

Fig. 4: Trace of the Client-2 and Server-2

will send an email based on the information in the graph to the mail server responsible for the destination domain, as soon as both peers agree with the full content of the tree. The fields take the normal semantics as used in every day's email traffic.

Results. To demonstrate the usability of the proposed method, both MailClient implementations were tested against both MailServer implementations. In all combinations the peers managed to reach agreement, while sticking to their internal business requirements as stated in table 1.

Figure 4 show the trace of Client-2 talking to Server-2. This is the most interesting combination due to the restrictions posed by the client. As can be seen in the first step of this figure, the client sets the value of the ToAddress.EmailAddress and subsequently applies the *ValidationRequest* operation to that node to communicate the request for validation. Since the server should support all operations, it replies appropriately to the request by using the *ValidationApprove* operation (second step). This satisfies the requirement of the client which as a result communicates the rest of the information through *SetValue* operations. The generic operations allow the client and server to interoperate even though the server has no advance knowledge regarding specific business requirements of this particular client.

5. Discussion

A use case with 4 client/server combinations was evaluated. The WSDL implementation demonstrated that unless future use is exceptionally well anticipated, different business requirements may easily lead to change requirements in the interface.

In contrast, for the Free Speech protocol the use case shows that there actually is room for local business requirements without the requirement to explicitly agree on these or to conform to these in advance. Since all the data ordering dependencies explicitly stem from the business requirements, any incompatibility that follows from these

dependencies is actually an incompatibility in the business requirements. Not being compatible on the business level is not something a protocol can solve, and is actually quite acceptable. However, other sources of dependencies have been taken out.

The range of application of the current specification is still limited. Currently only very few operations are supported, and those which are supported are tailored for this use case. Given the single use case it is hard to draw strong conclusions about the general usefulness of the Free Speech protocol. However, the current implementation does show that for a real-world application, this type of communication may indeed lead to successful communication. Future use cases will provide more experience and a complete set of operations supporting a wide variety of business requirements and interaction patterns without hindering compatibility.

A drawback of the current implementation is that the focus is completely on the low-level communication. The higher level workflow has been kept out of view here. This will also be addressed by future use cases, where different actions come into play on which a peer may require to put some ordering constraints.

An issue that also does not surface in the current use case, is the point of convergence. If the interaction is left unspecified, there is no guarantee that there will be convergence towards agreement. At least should the library be able to spot repetitions, and possibly support termination decision making for the peers.

A final issue that will be considered in the future is to add strong types to the vocabulary specification. Currently, all values are strings. While this has the advantage of not limiting the application in any way, it may also turn out to be too open in some cases. The option to restrict type of a value, either in the vocabulary or in the conversation itself will be investigated.

As argued, compared to the existing RFC paradigm, there are clear advantages in terms of the open nature of the protocol. However, there is also a downside. The Free Speech protocol is at best as efficient with respect to the communication, both in terms of time and bandwidth, but is in practice more to likely to be much less efficient as RFC based services. Also when regarding the implementation effort of peers, the RFC style is a winner provided that a suitable interface exists and that all other peers use that same interface. For closed environments these consideration may outweigh the advantages provided by Free Speech.

The use of operations has been inspired on the performatives used by the agents community. However, the type of operations and the fact that the 'subject' of the operation is restricted to specific nodes of a tree make that free speech really differs from the agent approach.

Finally the tuple/triple space approach has introduced a spatial and temporal disconnection that has been retained in the Free Speech protocol. In fact, the conceptual space with shared graphs is exactly what the Free Speech protocol evolves around. While triple spaces are only providing storage and retrieval facilities, the Free Speech protocol adds another layer of interaction rules (through the operations describing allowed updates to the trees) that allows a new level of interaction. In that sense Free Speech builds on top of Triple spaces, like for example WSDL builds on SOAP.

6. Conclusion

In this article, a new communication and interaction protocol for web services was introduced, called the Free Speech protocol. It is designed to overcome needless incompatibilities induced by the communication protocol as is the case with contemporary *Remote function call* style protocols like WSDL.

An initial prototype has been presented of a library – the Free Speech Engine – on top of which a simple set of client and servers was build. While the WSDL implementation required either changes to the interface, or a very good estimation beforehand of the intended future use to fit in all the business requirements of the peers, the Free Speech protocol allowed seamless interaction between all peers, regardless of their ignorance of each others business requirements and without the protocol enforcing specific business requirements.

Work needs to be done to mature the set of operations in such a way that also services with a rich set of business requirements is able to take full advantage of the system. Additionally, the high-level workflow perspective will be added to the equation. The initial effort looks promising and has offered useful insights and will be continued.

References

[1] David Booth and Canyang Kevin Liu. *"Web Services Description Language (WSDL) Version 2.0 — Part 0: Primer"*. Technical report, W3C, 2007. Available from: `http://www.w3.org/TR/wsdl20-primer/wsdl20-primer.pdf`.

[2] Brahim Chaib-draa and Frank Dignum. *Trends in Agent Communication Language*. *Computational Intelligence*, 18(2):89–101, 2002.

[3] The OWL Services Coalition. *"OWL-S: Semantic Markup for Web Services"*. Technical report, DAML, 2004. Available from: `http://www.daml.org/services/owl-s/1.1/overview/`.

[4] Daniel Elenius. *Modeling services with Protégé*. In *Proceedings of the 7th International Protégé Conference*, 2004.

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000.

[6] David Gelernter and Nicholas Carriero. *Coordination languages and their significance*. *Commun. ACM*, 35(2):97–107, 1992.

[7] Lyndon Nixon, Elena Simperl, Reto Krummenacher, and Francisco Martin-Recuerda. *Tuplespace-based Computing for the Semantic Web: A Survey of the State of the Art*. *Knowledge Engineering Review*, 23(2):181–212, 03 2008. Available from: `http://journals.cambridge.org/action/displayIssue?jid=KER&volumeId=23&issueId=02`.

[8] *Remote Method Invocation Home* [online]. Available from: `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp`.

[9] Dumitru Roman et al. *Web Service Modeling Ontology*. *Applied Ontology*, 1(1), 2005.

[10] *RPC: Remote Procedure Call Protocol Specification Version 2* [online]. Available from: `http://tools.ietf.org/html/rfc1831`.

[11] Elena Simperl, Reto Krummenacher, and Lyndon Nixon. *A Coordination Model for Triplespace Computing*, 4467 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007.

[12] Robert Tolksdorf, Elena Paslaru Bontas Simperl, and Lyndon J. B. Nixon. *Towards a tuplespace-based middleware for the Semantic Web*. *Journal of Web Intelligence*, 6(3):235–251, 2008.